Ex1 + ex2

GLfloat Vertices[] = {

// Cele 4 puncte din colturi;

-400.0f, -200.0f, 0.0f, 1.0f,

500.0f, -200.0f, 0.0f, 1.0f,

500.0f, 400.0f, 0.0f, 1.0f,

-400.0f, 400.0f, 0.0f, 1.0f,

// Varfuri pentru axe;

-400.0f, 0.0f, 0.0f, 1.0f,

500.0f, 0.0f, 0.0f, 1.0f,

0.0f, -200.0f, 0.0f, 1.0f,

0.0f, 400.0f, 0.0f, 1.0f,

};

GLfloat Colors[] = {

1.0f, 0.0f, 0.0f, 1.0f,

0.0f, 1.0f, 0.0f, 1.0f,

0.0f, 0.0f, 1.0f, 1.0f,

0.8f, 0.4f, 0.0f, 1.0f,

1.0f, 0.0f, 0.0f, 1.0f,

0.0f, 1.0f, 0.0f, 1.0f,

0.0f, 0.0f, 1.0f, 1.0f,

0.8f, 0.4f, 0.0f, 1.0f,

};

codCol = 0; // Culoarea;

glUniform1i(codColLocation, codCol); // Transmiterea variabilei uniforme pentru COLORARE spre shadere;

glPointSize(15.0); // Se seteaza dimensiunea punctelor si netezirea marginilor;

glEnable(GL\_POINT\_SMOOTH);

// Functia de desenare primeste 3 argumente:

// - arg1 = tipul primitivei desenate,

// - arg2 = indicele primului punct de desenat din buffer,

// - arg3 = numarul de puncte consecutive de desenat;

glDrawArrays(GL\_POLYGON, 0, 4);

// Desenarea axelor;

codCol = 1; // Culoarea;

glUniform1i(codColLocation, codCol);

glLineWidth(2.0);

glDrawArrays(GL\_LINES, 4, 4);

// Desenarea dreptunghiul;

// Matricea asociata dreptunghiul - este translatat fata de axe;

codCol = 2; // Culoarea;

myMatrix = resizeMatrix \* matrTransl; // ATENTIE LA ORDINE!

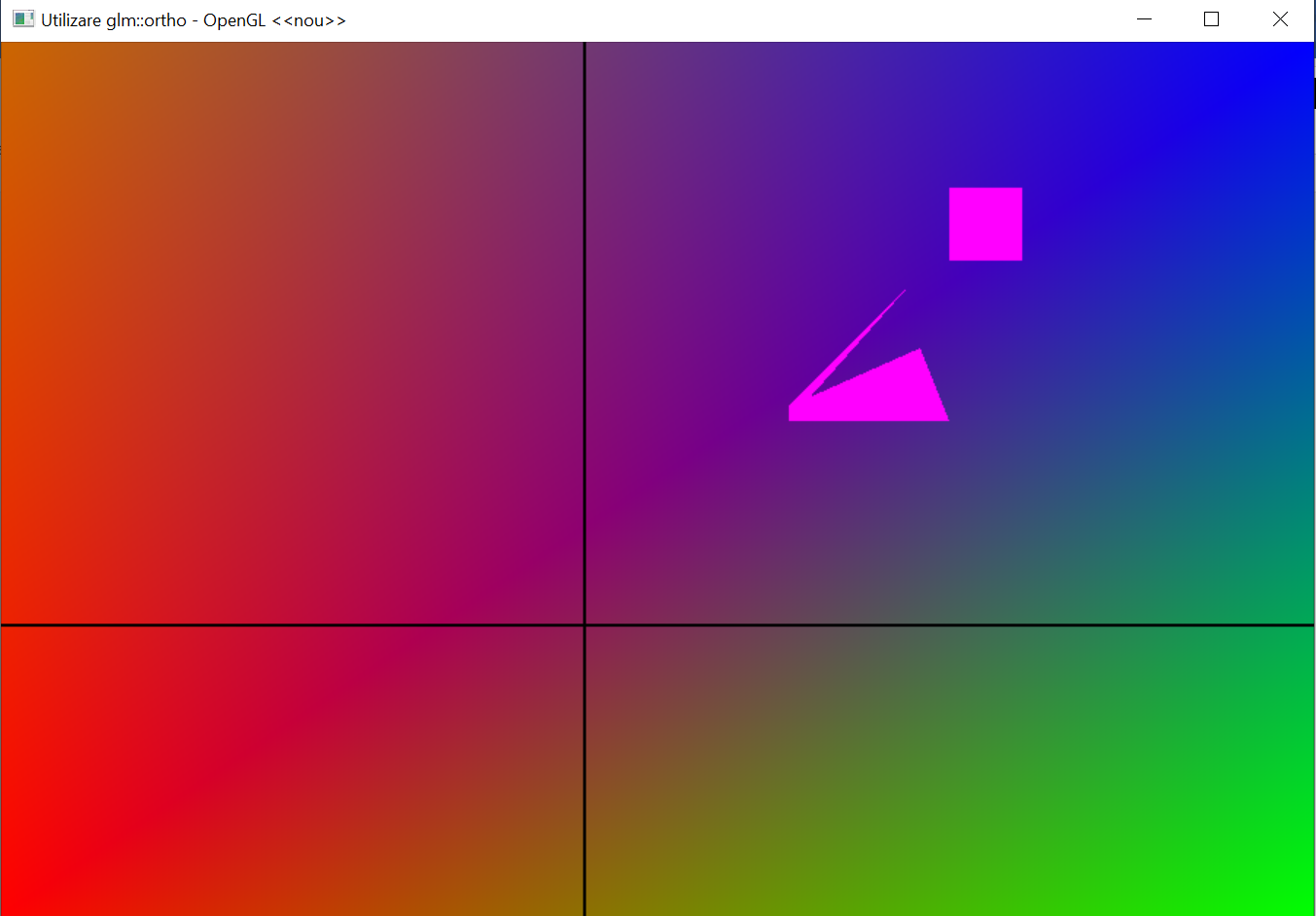
// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE si COLORARE spre shadere;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_TRIANGLE\_FAN, 8, 4);

glDrawArrays(GL\_TRIANGLE\_FAN, 12, 5);



3. GLfloat Vertices[] = {

// Varfuri pentru axe;

-400.0f, 0.0f, 0.0f, 1.0f,

400.0f, 0.0f, 0.0f, 1.0f,

0.0f, -300.0f, 0.0f, 1.0f,

0.0f, 300.0f, 0.0f, 1.0f,

// // Varfuri pentru dreptunghi;

//50.0f, 50.0f, 0.0f, 1.0f,

//50.0f, 150.0f, 0.0f, 1.0f,

//150.0f, 150.0f, 0.0f, 1.0f,

//150.0f, 50.0f, 0.0f, 1.0f,

// Varfuri pentru poligon convex;

150.0f, 130.0f, 0.0f, 1.0f,

300.0f, 130.0f, 0.0f, 1.0f,

300.0f, 290.0f, 0.0f, 1.0f,

150.0f, 190.0f, 0.0f, 1.0f,

// Varfuri poligon concav;

50.0f, 50.0f, 0.0f, 1.0f,

150.0f, 70.0f, 0.0f, 1.0f,

80.0f, 100.0f, 0.0f, 1.0f,

45.0f, 170.0f, 0.0f, 1.0f,

};

// Culorile axelor;

GLfloat Colors[] = {

1.0f, 0.0f, 0.0f, 1.0f,

0.0f, 1.0f, 0.0f, 1.0f,

0.0f, 0.0f, 1.0f, 1.0f,

1.0f, 0.0f, 0.0f, 1.0f,

1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

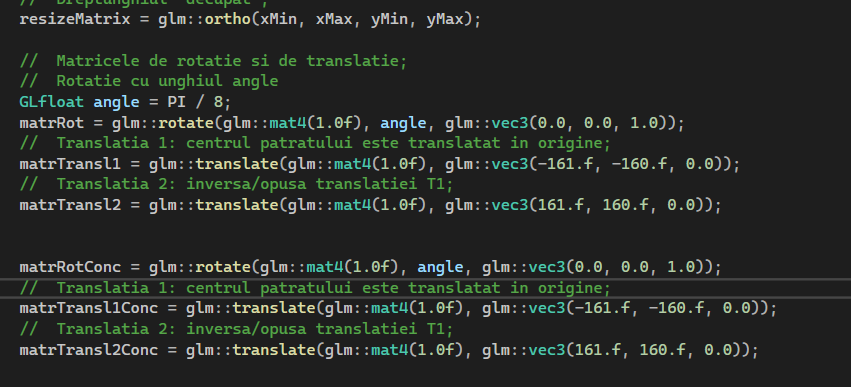
1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

1.0f, 0.0f, 1.0f, 0.0f,

};



glClear(GL\_COLOR\_BUFFER\_BIT); // Se curata ecranul OpenGL pentru a fi desenat noul continut;

// Matrice pentru elemente "fixe" - axe;

// Transmiterea variabilei uniforme pentru MATRICEA DE TRANSFORMARE spre shadere;

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

// Desenarea axelor;

codCol = 0; // Culoarea;

glUniform1i(codColLocation, codCol); // Transmiterea variabilei uniforme pentru COLORARE spre shadere;

glLineWidth(1.0); // Se seteaza dimensiunea liniilor;

// Functia de desenare primeste 3 argumente:

// - arg1 = tipul primitivei desenate,

// - arg2 = indicele primului punct de desenat din buffer,

// - arg3 = numarul de puncte consecutive de desenat;

glDrawArrays(GL\_LINES, 0, 4);

// Desenarea poligon cob initial;

codCol = 1; // Culoare;

glUniform1i(codColLocation, codCol);

glLineWidth(2.0);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL);

glDrawArrays(GL\_QUADS, 4, 4);

// Desenarea dreptunghiului desenat;

// Matricea pentru rotire cand poligonul nu are centrul in origine:

// - se translateaza in origine,

// - se roteste,

// - se translateaza inapoi in pozitia initiala;

// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE si COLORARE spre shadere;

// Transmiterea variabilelor uniforme pentru MATRICEA DE TRANSFORMARE si COLORARE spre shadere;

glDrawArrays(GL\_TRIANGLE\_FAN, 8, 4);

codCol = 2;

myMatrix = resizeMatrix \* matrTransl2Conc \* matrRotConc \* matrTransl1Conc;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glDrawArrays(GL\_QUADS, 8, 4);

codCol = 2;

myMatrix = resizeMatrix \* matrTransl2 \* matrRot \* matrTransl1;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

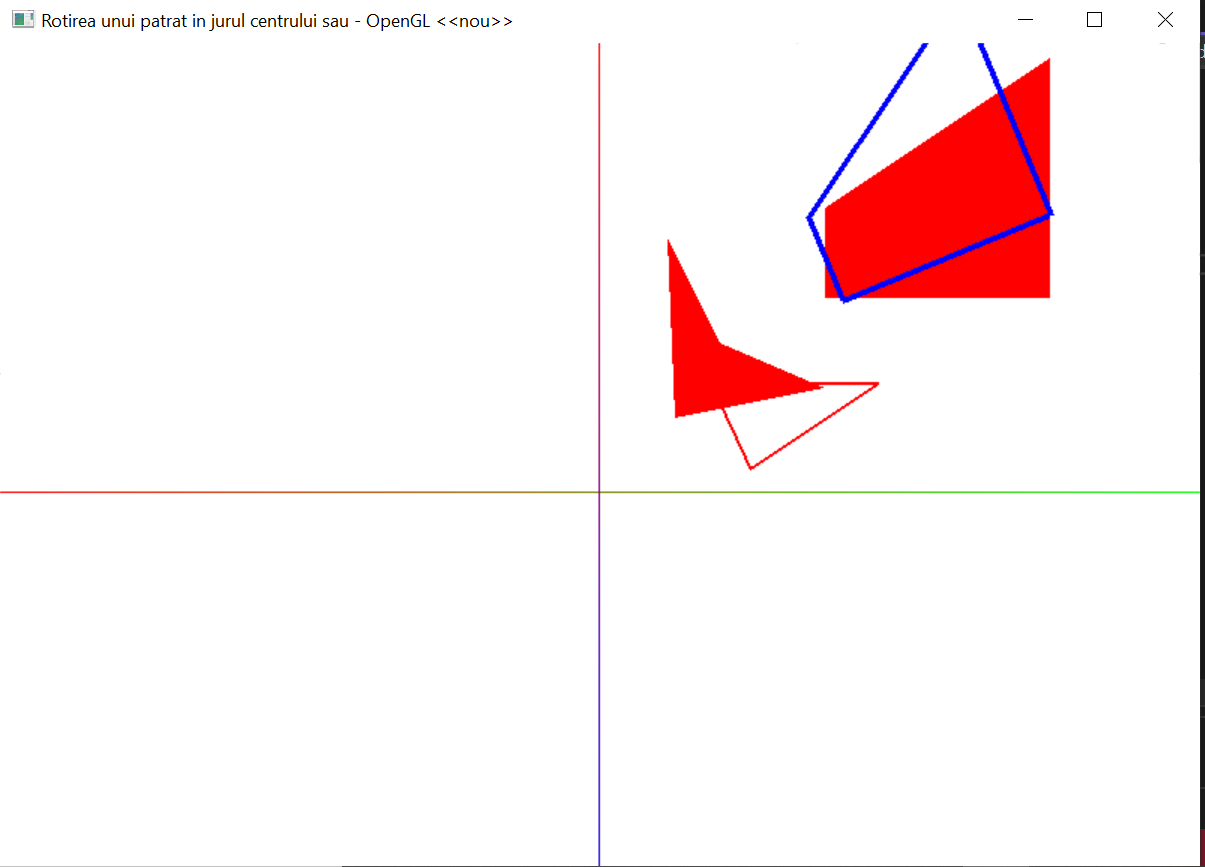
glUniform1i(codColLocation, codCol);

glLineWidth(4.0);

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE);

glDrawArrays(GL\_QUADS, 4, 4);

glFlush();



4.

//

// ================================================

// | Grafica pe calculator |

// =====================================================

// | Laboratorul III - 03\_05\_transformari\_keyboard.cpp |

// =====================================================

//

// Realizarea unei scene 2D in care obiectele se misca, folosindu-se tehnicile MODERN OpenGL;

// ELEMENTE DE NOUTATE:

// - functii pentru tastatura;

//

//

//

// Biblioteci

#include <windows.h> // Utilizarea functiilor de sistem Windows (crearea de ferestre, manipularea fisierelor si directoarelor);

#include <stdlib.h> // Biblioteci necesare pentru citirea shaderelor;

#include <stdio.h>

#include <GL/glew.h> // Definește prototipurile functiilor OpenGL si constantele necesare pentru programarea OpenGL moderna;

#include <GL/freeglut.h> // Include functii pentru:

// - gestionarea ferestrelor si evenimentelor de tastatura si mouse,

// - desenarea de primitive grafice precum dreptunghiuri, cercuri sau linii,

// - crearea de meniuri si submeniuri;

#include "loadShaders.h" // Fisierul care face legatura intre program si shadere;

#include "glm/glm.hpp" // Bibloteci utilizate pentru transformari grafice;

#include "glm/gtc/matrix\_transform.hpp"

#include "glm/gtx/transform.hpp"

#include "glm/gtc/type\_ptr.hpp"

// Identificatorii obiectelor de tip OpenGL;

GLuint

VaoId,

VboId,

ColorBufferId,

ProgramId,

myMatrixLocation,

matrScaleLocation,

matrTranslLocation,

matrRotlLocation,

codColLocation;

// Dimensiunile ferestrei de afisare;

GLfloat

winWidth = 800, winHeight = 600;

// Variabile catre matricile de transformare;

glm::mat4

myMatrix, resizeMatrix, matrTransl, matrScale, matrRot;

int codCol; // Variabila ce determina schimbarea culorii pixelilor in shader;

float angle = 0; // Unghiul de rotire al patratului;

float tx = 0; float ty = 0; float auxtx; float auxangle; // Coordonatele de translatie ale patratului pe Ox si Oy;

float xMin = -400.f, xMax = 400.f, yMin = -300.f, yMax = 300.f; // Variabile pentru proiectia ortogonala;

void ProcessNormalKeys(unsigned char key, int x, int y)

{

switch (key) { // Procesarea tastelor 'l' si 'r' modifica unghiul de rotire al patratului;

case 'l':

angle += 0.2f; // Rotire spre stanga;

break;

case 'r':

angle -= 0.2f; // Rotire spre dreapta;

break;

}

if (key == 27)

exit(0);

}

void ProcessSpecialKeys(int key, int xx, int yy)

{

switch (key) // Procesarea tastelor 'LEFT', 'RIGHT', 'UP', 'DOWN'

{ // duce la deplasarea patratului pe axele Ox si Oy;

case GLUT\_KEY\_LEFT:

tx -= 10;

angle += 0.2f;

break;

case GLUT\_KEY\_RIGHT:

tx += 10;

angle -= 0.2f;

break;

case GLUT\_KEY\_UP:

ty += 10;

angle += 0.2f;

break;

case GLUT\_KEY\_DOWN:

ty -= 10;

angle -= 0.2f;

break;

}

}

// Crearea si compilarea obiectelor de tip shader;

// Trebuie sa fie in acelasi director cu proiectul actual;

// Shaderul de varfuri / Vertex shader - afecteaza geometria scenei;

// Shaderul de fragment / Fragment shader - afecteaza culoarea pixelilor;

void CreateShaders(void)

{

ProgramId = LoadShaders("03\_05\_Shader.vert", "03\_05\_Shader.frag");

glUseProgram(ProgramId);

}

// Se initializeaza un Vertex Buffer Object (VBO) pentru tranferul datelor spre memoria placii grafice (spre shadere);

// In acesta se stocheaza date despre varfuri (coordonate, culori, indici, texturare etc.);

void CreateVBO(void)

{

// Coordonatele varfurilor;

GLfloat Vertices[] = {

// Cele 4 varfuri din colturi;

-390.0f, -290.0f, 0.0f, 1.0f,

390.0f, -290.0f, 0.0f, 1.0f,

390.0f, 290.0f, 0.0f, 1.0f,

-390.0f, 290.0f, 0.0f, 1.0f,

// Varfuri pentru axe;

-400.0f, 0.0f, 0.0f, 1.0f,

400.0f, 0.0f, 0.0f, 1.0f,

// Varfuri pentru dreptunghi;

-20.0f, -20.0f, 0.0f, 1.0f,

20.0f, -20.0f, 0.0f, 1.0f,

20.0f, 20.0f, 0.0f, 1.0f,

-20.0f, 20.0f, 0.0f, 1.0f,

// Originea;

0.0f, 0.0f, 0.0f, 1.0f

};

// Culorile punctelor din colturi;

GLfloat Colors[] = {

1.0f, 1.0f, 0.0f, 1.0f,

0.0f, 1.0f, 1.0f, 1.0f,

1.0f, 0.0f, 1.0f, 1.0f,

0.5f, 0.5f, 0.1f, 1.0f,

};

// Transmiterea datelor prin buffere;

// Se creeaza / se leaga un VAO (Vertex Array Object) - util cand se utilizeaza mai multe VBO;

glGenVertexArrays(1, &VaoId); // Generarea VAO si indexarea acestuia catre variabila VaoId;

glBindVertexArray(VaoId);

// Se creeaza un buffer pentru VARFURI;

glGenBuffers(1, &VboId); // Generarea bufferului si indexarea acestuia catre variabila VboId;

glBindBuffer(GL\_ARRAY\_BUFFER, VboId); // Setarea tipului de buffer - atributele varfurilor;

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Vertices), Vertices, GL\_STATIC\_DRAW); // Punctele sunt "copiate" in bufferul curent;

// Se asociaza atributul (0 = coordonate) pentru shader;

glEnableVertexAttribArray(0);

glVertexAttribPointer(0, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

// Se creeaza un buffer pentru CULOARE;

glGenBuffers(1, &ColorBufferId);

glBindBuffer(GL\_ARRAY\_BUFFER, ColorBufferId);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(Colors), Colors, GL\_STATIC\_DRAW);

// Se asociaza atributul (1 = culoare) pentru shader;

glEnableVertexAttribArray(1);

glVertexAttribPointer(1, 4, GL\_FLOAT, GL\_FALSE, 0, 0);

}

// Elimina obiectele de tip shader dupa rulare;

void DestroyShaders(void)

{

glDeleteProgram(ProgramId);

}

// Eliminarea obiectelor de tip VBO dupa rulare;

void DestroyVBO(void)

{

// Eliberarea atributelor din shadere (pozitie, culoare, texturare etc.);

glDisableVertexAttribArray(1);

glDisableVertexAttribArray(0);

// Stergerea bufferelor pentru varfuri, culori;

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glDeleteBuffers(1, &ColorBufferId);

glDeleteBuffers(1, &VboId);

// Eliberaea obiectelor de tip VAO;

glBindVertexArray(0);

glDeleteVertexArrays(1, &VaoId);

}

// Functia de eliberare a resurselor alocate de program;

void Cleanup(void)

{

DestroyShaders();

DestroyVBO();

}

// Setarea parametrilor necesari pentru fereastra de vizualizare;

void Initialize(void)

{

glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // Culoarea de fond a ecranului;

CreateVBO(); // Trecerea datelor de randare spre bufferul folosit de shadere;

CreateShaders(); // Initilizarea shaderelor;

// Instantierea variabilelor uniforme pentru a "comunica" cu shaderele;

codColLocation = glGetUniformLocation(ProgramId, "codCol");

myMatrixLocation = glGetUniformLocation(ProgramId, "myMatrix");

// Dreptunghiul "decupat";

resizeMatrix = glm::ortho(xMin, xMax, yMin, yMax);

}

// Functia de desenarea a graficii pe ecran;

void RenderFunction(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT); // Se curata ecranul OpenGL pentru a fi desenat noul continut;

// Matrici pentru transformari;

matrTransl = glm::translate(glm::mat4(1.0f), glm::vec3(tx, ty, 0.0));

matrRot = glm::rotate(glm::mat4(1.0f), angle, glm::vec3(0.0, 0.0, 1.0));

// Matrice pentru elemente "fixe" - axe;

// Transmiterea variabilei uniforme pentru MATRICEA DE TRANSFORMARE spre shadere;

myMatrix = resizeMatrix;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

// Desenarea punctelor din colturi si a axelor;

codCol = 0; // Culoarea;

glUniform1i(codColLocation, codCol); // Transmiterea variabilei uniforme pentru COLORARE spre shadere;

glPointSize(10.0); // Se seteaza dimensiunea punctelor;

// Functia de desenare primeste 3 argumente:

// - arg1 = tipul primitivei desenate,

// - arg2 = indicele primului punct de desenat din buffer,

// - arg3 = numarul de puncte consecutive de desenat;

glDrawArrays(GL\_POINTS, 0, 4);

glDrawArrays(GL\_LINES, 4, 2);

// Matricea pentru elementele care isi schimba pozitia;

myMatrix = resizeMatrix \* matrTransl \* matrRot;

glUniformMatrix4fv(myMatrixLocation, 1, GL\_FALSE, &myMatrix[0][0]);

// Desenare patrat si punct mobil;

codCol = 1; // Schimbare culoare;

glUniform1i(codColLocation, codCol);

glDrawArrays(GL\_POLYGON, 6, 4);

codCol = 2; // Schimbare culoare;

glUniform1i(codColLocation, codCol);

glEnable(GL\_POINT\_SMOOTH); // Netezirea marginilor punctelor;

//glDrawArrays(GL\_POINTS, 10, 1);

glDisable(GL\_POINT\_SMOOTH);

glutSwapBuffers(); // Inlocuieste imaginea deseneata in fereastra cu cea randata;

glFlush(); // Asigura rularea tuturor comenzilor OpenGL apelate anterior;

}

// Punctul de intrare in program, se ruleaza rutina OpenGL;

int main(int argc, char\* argv[])

{

// Se initializeaza GLUT si contextul OpenGL si se configureaza fereastra si modul de afisare;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB); // Se folosesc 2 buffere (unul pentru afisare si unul pentru randare => animatii cursive) si culori RGB;

glutInitWindowSize(winWidth, winHeight); // Dimensiunea ferestrei;

glutInitWindowPosition(100, 100); // Pozitia initiala a ferestrei;

glutCreateWindow("Utilizarea tastaturii - OpenGL <<nou>>"); // Creeaza fereastra de vizualizare, indicand numele acesteia;

// Se initializeaza GLEW si se verifica suportul de extensii OpenGL modern disponibile pe sistemul gazda;

// Trebuie initializat inainte de desenare;

glewInit();

Initialize(); // Setarea parametrilor necesari pentru fereastra de vizualizare;

glutDisplayFunc(RenderFunction); // Desenarea scenei in fereastra;

glutIdleFunc(RenderFunction); // Asigura rularea continua a randarii;

glutKeyboardFunc(ProcessNormalKeys); // Functii ce proceseaza inputul de la tastatura utilizatorului;

glutSpecialFunc(ProcessSpecialKeys);

glutCloseFunc(Cleanup); // Eliberarea resurselor alocate de program;

// Bucla principala de procesare a evenimentelor GLUT (functiile care incep cu glut: glutInit etc.) este pornita;

// Prelucreaza evenimentele si deseneaza fereastra OpenGL pana cand utilizatorul o inchide;

glutMainLoop();

return 0;

}